

Number Theory (continued), Public Key Cryptography

Continuing from the last lecture – finding an inverse:

We want to find the inverse of a number a in $(\mathbb{Z} / n\mathbb{Z})^*$

i.e. given a , we want to find b such that $a * b \equiv 1 \pmod{n}$.

Now, $ab \equiv 1 \pmod{n} \iff ab = kn + 1$ for some k

i.e. $ab - kn = 1$

We can find b (and k) using the Euclidean algorithm.

Euclidean algorithm:

The following equations hold true:

$$1 \times a + 0 \times n = a \quad (1)$$

$$0 \times a + 1 \times n = n \quad (2)$$

From the definition of $(\mathbb{Z} / n\mathbb{Z})^*$ we know that $a < n$.

We now find a value m_1 such that $m_1 a \leq n$

but $(m_1 + 1) a > n$

So $m_1 a$ is the largest multiple of a that is less than n .

i.e. $m_1 = \lfloor n / a \rfloor$

Multiplying equation (2) by m_1 and subtracting the result from equation (1), we have

$$-m_1 a + 1 \cdot n = n - m_1 a$$

Say $d \mid a$ and $d \mid n$ [$x \mid y$ indicates that x is a divisor of y]

If d is a divisor of a and n , then d is also a divisor of $n - m_1 a$

i.e. $d \mid (n - m_1 a)$

Fact 1: Every divisor of a and n is a divisor of $(n - m_1a)$

[See sidebar for more.]

So in the Euclidean algorithm, the set of divisors of the right hand side of the equations is preserved at every step.

Fact 2: The value on the RHS of the equations gets smaller at each step.

Fact 3: Eventually, only the divisors will be left on the RHS. Then, the RHS is the gcd of (a, n) .

All equations are of the form:

$$x.a + y.n = r$$

Eventually, r will become the gcd of a and n .

i.e. it will become 1 for $a \in (\mathbb{Z} / n\mathbb{Z})^*$

Therefore, we will get $x.a + y.n = 1$

Example:

We want to find the inverse of 37, modulo 61.

i.e. $37^{-1} \pmod{61}$

$$1 \times 37 + 0 \times 61 = 37 \tag{1}$$

$$0 \times 37 + 1 \times 61 = 61 \tag{2}$$

$$m_1 = \lfloor 61 / 37 \rfloor = 1$$

By (2) - m_1 (1) we get:

$$-1 \times 37 + 1 \times 61 = 24 \tag{3}$$

Now consider (1) and (3).

$$m_2 = \lfloor 37 / 24 \rfloor = 1$$

By (1) - m_2 (3) we get:

$$2 \times 37 - 1 \times 61 = 13 \tag{4}$$

Sidebar:

We can also show that every divisor of a and $(n - m_1a)$ must be a divisor of n :

Consider:

$$n - m_1a = xd \quad \text{for some } x$$

$$\text{i.e. } n - m_1yd = xd \quad \text{for some } y$$

(since both a and $n - m_1a$ are some multiple of d)

Then,

$$n = xd + m_1yd$$

$$= d(x + m_1y)$$

That is, n is also a multiple of d .

Now consider (3) and (4)

$$m_3 = \lfloor 24 / 13 \rfloor = 1$$

By (3) – m_3 (4) we get:

$$-3 \times 37 + 2 \times 61 = 11 \quad (5)$$

Now consider (4) and (5)

$$m_4 = \lfloor 13 / 11 \rfloor = 1$$

By (4) – m_4 (5) we get:

$$5 \times 37 - 3 \times 61 = 2 \quad (6)$$

Now with (5) and (6)

$$m_5 = \lfloor 11 / 2 \rfloor = 5$$

By (5) – m_5 (6) we get:

$$-28 \times 37 + 17 \times 61 = 1 \quad (7)$$

We now have 1 on the RHS.

(7) is of the form $b \times a - k \times n = 1$. Therefore, $b = -28$

A negative number (-28) is not acceptable as the inverse. We add 61 (i.e. keep adding n until we get a positive number) to get 33.

$$\text{Thus } 37 * 33 \equiv 1 \pmod{61}$$

Efficient algorithm for computing exponentiation modulo n :

We need a way to compute $a^b \pmod{n}$

where b is huge (for instance, 2000 bits)

To do this, we write b in binary.

$$\begin{aligned} \text{eg. } b &= 1001 \\ &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 2^3 + 1 \times 2^0 \end{aligned}$$

Then we can write:

$$\begin{aligned} a^b &= a^{2^3+2^0} && [x^y \text{ indicates } x^y] \\ &= a^{2^3} \times a^{2^0} \end{aligned}$$

So we can compute the exponentiation as follows:

$$\begin{aligned} a^{2^0} &= a & 1 \\ a^{2^1} &= a^2 & 0 \\ a^{2^2} &= a^4 & 0 \\ a^{2^3} &= a^8 & 1 \end{aligned}$$

i.e. calculate product of bits of b with the corresponding exponent of a , and then compute the product of these sub-products. In the above case, it involves a maximum of 3 squaring operations and 3 multiplication operations.

In general, if b is n bits long, this method requires a maximum of $n - 1$ squaring operations and $n - 1$ multiplication operations.

Note: The individual exponents of a can also be stored modulo n because in general,
 $xy \bmod n = [(x \bmod n) * (y \bmod n)] \bmod n$

Algorithm: to compute $a^b \bmod n$

1. Compute $a^{2^i} \bmod n$
for $i = 0, 1, \dots, |b|$
2. Take product of all $a^{2^i} \bmod n$ where $b_i = 1$
i.e. compute $\prod_{b_i=1} a^{2^i} \bmod n$

Using the mathematical background discussed, we can now define the RSA public-key cryptosystem.

RSA Public-key Cryptosystem:

Key Generation:

- Alice selects two large primes p and q . She then computes their product.
 $N = p \times q$
- Alice then selects a number e such that $\gcd(e, \phi(N)) = 1$
- Alice then computes d such that $e*d \equiv 1 \pmod{\phi(N)}$

[Note that as defined earlier, $\phi(N) = (p - 1)(q - 1)$]

Then Alice's public key = (N, e)
 and Alice's private key = (N, d)

Encryption:

The encryption is done using the public key.
 Let the message to be encrypted be $M \in (\mathbb{Z} / n\mathbb{Z})^*$
 The ciphertext C is computed as

$$C = M^e \pmod N$$

Decryption:

The decryption is done using the private key.

$$M = C^d \pmod N$$

$$\begin{aligned} \text{Because } C^d \pmod N &= M^{ed} \pmod N \\ &= M^{k\varphi(N)+1} \pmod N && [\text{Because } e*d \equiv 1 \pmod{\varphi(N)}] \\ &= M^{(\varphi(N))^k} \pmod N \\ &= 1^k \cdot M \pmod N = M && [\text{Theorem: For any finite group } G \\ &&& \text{and } a \in G, a^{|G|} = 1] \end{aligned}$$

The most obvious attack on RSA is that a malicious user can try to find d from (N, e) , i.e. try to get the private key from the public key. However, it can be shown that if the attacker can find d from (N, e) , then the attacker can factor N .
(Proof is a homework problem.)

Therefore, N must be hard to factor.
 For this, both p and q must be of about the same size.
 Thumb rule: $p, q \approx$ square root of N

Also, N should be large. But how large?

- 300 bits can be factored in seconds on a PC
- 512 bits can be factored with a small cluster.
- 768 bits is conjectured to be factorable with \$10 million

So we should make N significantly larger than this.
 Today, usually $N \approx 2048$ bits.

Strong RSA Assumption:

Given N, C , it is computationally infeasible to find M, e such that $M^e \equiv C \pmod N$.

The weaker assumption says that it is computationally infeasible to find M given C and N for a specific key pair (i.e. a specific pair of e and d). The stronger assumption lets the attacker choose e as well, i.e. it says that it is computationally infeasible to find *any* pair of (M, e) given C and N .

Common implementation practice:

In general, RSA is significantly more computationally intensive than symmetric key algorithms, because RSA involves a large number of exponentiation operations on large numbers.

Therefore, to make things easier, in practice, usually $e = 3$.

So encrypting M only requires doing M^3 , i.e. one square and one multiplication. In this case, encryption is fast but decryption is slow.

(As $e = 3$, the value of d will be significantly larger. And despite a constant e , the value of d will be different for different values of N .)

It is inefficient to encrypt a large file completely using RSA. Therefore, a common practice to encrypt a large file M is as follows:

- Pick a key k
- Encrypt the file using any block cipher with the key k
 $C_0 = E_k(M)$
- Encrypt the key k with the public key of the intended recipient.
 $C_1 = \text{RSA}_{\text{Pub}}(k)$
- Transmit C_0C_1 as the ciphertext.

Then, to decrypt, the recipient would have to decrypt C_1 with her private key to obtain the key k and then decrypt C_0 using that key.

* * *